**软件需求分析与建模心得**

华亮 200930634478

写这份心得的时候，发现已经离项目暂时完工已经很长时间了。或许感受也没有之前那么深刻。

这学期从开学来到学校就一直忙着做师兄的项目，无暇听课和做自己的大作业。但是从师兄和师兄的师兄那里学到很多很多实际的项目开发经验。可惜的是，项目是一个失败的结局，在大二第一学期和寒假，项目基本没有动工，我作为其中一个成员，没有积极参与开发。而管理者也明显具有失误，没有调用团队的积极性。以至于项目的开发都基本集中到截止时间的最后一个月才开始疯狂赶工而最后也只能草草收尾。

鉴于原来的项目痛心的失败经验，我深刻明白了一个团队的领导者是如此的重要。一个没有领导的团队就是一盘散沙，大家朝着不同的方向前进着，最后整个团队还是在原点。

一个团队的领导者首先要让团队里的每一个人成员知道自己要做什么，为什么要做。

经历过的结局不太好的项目发现有几个原因。第一点是，自己不知道自己现在要做什么，以至于自以为是的认为自己一下就可以完成而往后推。但实际最后去做的时候发现根本做不完。第二个失败点是自己不知道要去做什么而为了面子又不愿向“上头”请教。

而之前自己独自做的拖控件生成配置文件的软件，实验室的老师、师兄的要求在不断变更，程序一改再改，曾一段时间恼怒不已，心中大喊一个软件变来变去干啥，为什么不一次性讲清楚！

第一次C++/MFC实现，为了速度几天就写了几千行代码，看似好像完成了，但最后修改的时候发现根本无法下手，自己都无法理解自己当初为什么这样写。程序已经变得不可维护了。第二次和师兄参加其他活动，就把那个软件弄到另一个项目中，改用C#/WPF实现，这样做的界面可以更轻易地做得很炫。WPF的从人的角度的高度抽象的软件开发更像是在组装一个程序，在用WPF开发界面后，对自己的建模设计能力有了一定的锻炼。第三次是这学期，当准备开始做自己的大作业时，实验室的软件需求变更再次来临时，我已经深刻体会到不认真的设计、不完备的思考的记录，最后必然导致举步维艰的维护。于是乎按照自己非系统学习过的乱七八糟的UML建模语言还是认真设计。最初在做这个程序时，不明白他有什么用，不知要该有什么功能，就草草开始了开发。很庆幸这学期开了《软件需求分析设计与建模》，当时讲课才讲到用例，也是第一次听到了“用例”这个名词，以前在书上见到的都是些零散的UML的类图，就傻傻地认为建模就是将类与类之间的继承关系描述出来时就好，而非继承关系的类之间的关系的描述，我也不知道如何描述，就纯文字再按照自己的想法画了些图。曾经有自己去看UML，但是最后还是不了了之。于是乎按照《UML和模式应用》中的用例分析，给实验室的软件写了许多用例，假象在各种场景下用户（老师和师兄）会怎样用，着实为自己找到了一些功能需求。

3月底，和师兄做的项目已经Deadline，实验室的软件也很快重写了。于是有时间实现自己的想法了。

为什么会有这款软件?

数据结构课上，老师给我们展示了一个很老的清华大学的数据结构演示Demo，有代码有动画，章节详细，是一个很好的数据结构教学软件。不过总觉得这款软件缺少什么。这款软件不能在上面写代码，只能看，不能动手，而且还有许多代码是伪代码，于是有人评论充其量不就是个Flash吧。确实，这个用Flash来做不就差不多嘛，不过年代比较久远，可能那时Flash还刚刚问世。于是我们想如果可以在里面自己写代码那就完美了！

中国学生缺的就是自己动手实践的能力，而且数据结构那么抽象，如果可以看到自己代码编写出来的数据结构的图形化形式，就能更好的理解数据结构了。于是就果断把这个作为数据结构大作业。当时组内只有杨旭瑜、罗嘉飞和我，后来罗嘉飞同学又继续拉上了安迪和杨明锦。于是我们组5人就一起上了贼船！

听杨旭瑜同学说道，广工的数据结构教学时学生写作业用的就是一款可以自己写代码而且可以显示数据结构。于是旭瑜和我专程去了广工看了广工的由老师带队做的数据结构写作业软件（不知如何称呼他们的软件）。他们的软件确实可以自己写代码，编译，而且也有图形化的数据结构。不过他们的专题性很强，要先选择一个数据结构的主题，然后程序会提示已经定义好的结构体是什么，函数的传入参数是什么。然后学生能在上面写的只有一个函数，例如，插入链表，就是只能写一个插入函数，而且函数原型系统都帮学生定义好了，传入一个链表的头结点和一个元素。学生自己不能改函数的定义，学生只能在这个函数中填入自己的代码，其他代码看不到。这样的限制太死了！一个程序绝不仅仅只是一个函数，这样的软件只会教坏学生！实践一个数据结构，自己竟然不能定义数据结构ADT，写程序就像在做填空题！不禁为广工的数据结构课程感到惋惜……使用了一会儿广工的软件后，也大概猜测出他的工作原理。不过他们的软件因为学生只能写一个函数，所以实现起来会比较容易，但对用户是不友好的。

于是构想，可以让用户把自己平时在VS、GCC里面编译过的代码直接粘贴到我们的程序上，就可以以图像的形式看到他自己编写的数据结构，这样更方便用户，也让用户更加真切地体会到真实的软件开发，而不是中国式的应试软件学习。

一开始觉得这样实现技术难度感觉很大，而且没什么头绪。于是想试验一下自己的想法，就花了一个晚上写了个《编译原理》课本讲的Tiny语言的基于语法树的解释器（因为那时编译原理还没看到代码生成，只懂语法分析，所以就这样实现），也实现了单步执行和运行时的变量信息显示。于是发现这样是可行的，就和罗嘉飞一起开始弄C语言的编译器，也开始将Tiny的语法树解释器修改为C语言的语法树解释器。鉴于C语言过于复杂，我们从中选取了足以描述算法和数据结构的语法，放弃实现一些语法，将C语言的文法进行了精简。就开始做自顶向下的语法分析器。

整个程序的架构还是比较经典的三层架构/MVC，界面层，控制器层和模型层。最底层中的编译器子系统也是很老的pipes and filters架构。整个程序析取了Microkernel架构，只想把最核心的功能做出来，其他的功能作为插件插入到系统中。而最核心的就是其中的编译器和虚拟机。其他的动画、内存信息、输出等等都应该作为插件。

最初做开发时用的界面是命令行的界面，和一个简易的图像界面，可以随意选择其中一个界面。程序的建模重点也主要在编译器和语法树解释器上。

后来编译原理学习到了代码生成，发现一个汇编虚拟机会比语法树解释器通用性更强。于是就改成先将C语言翻译成汇编，然后让汇编在汇编虚拟机中运行。这样的设计就可以让多种语言只要翻译成统一的汇编即可在虚拟机上运行。如此设计的系统的扩展性大大提高，也方便日后的系统扩充。

在之前的程序中，有两套词法器和语法器，为了切换方便就将语法器和词法器的抽象基类抽取出来，确实，面向对象中抽象、封装、继承、多态的思想，有时会帮助你去解决一些问题。

在动画模块的设计上，直接第一感觉就是工厂模式可以很好的解决这个问题。因为之前实践过，也在以前的项目中见过牛人师兄用来解决过类似的问题。后来证实，确实很好地完成的动画模块的设计。

而我很久以前留下的一个不知如何描述非继承的类之间关系的问题，终于在课上得到的答案。原来除了静态图，更重要的是像顺序图、活动图之类的动态图，以描述对象与对象间的协作关系，这样就可以很好地将自己的思想记录下来与人分享，也可以方便自己思考问题。

对于需求分析，在大一时无法理解何为需求分析这个词语是什么意思，不过上完这门课也总算有了个大概的认识。现在我们大多数开发的项目基本是需求都是没什么变更的，这时软件设计的好与坏可能并不那么容易体现出来。而真正的软件，他的需求是不断变更的。这时，就要求我们设计的软件能够欢迎变更，拥抱变更！我们需要花更多的精力去设计更加优美更加灵活的架构。

需求，用例描述着需求，用例驱动着软件开发。

架构是一个程序的骨架，设计得不好必然导致程序的畸形发展。现有的许多架构确实很优秀，但是有时他们真的适合我们吗？是生搬硬套像革命年代部分人拘泥于马克思主义的具体纲领，还是像毛泽东一样理论与实际相结合？

对于这门课的看法，对于有大量的项目经验的同学，这门课是提升自己重要课程；对于有一定项目经验的同学，这门课也是软件工程的第一扇门；对于缺少项目经验的，这门课沦入了毛概马哲之列。

不知为何在软件工程创建之初，软件工程是面向已经工作4年的程序员……